STRUCTURED COMPUTER ORGANIZATION

ANDREW S. TANENBAUM

Vrije Universiteit
Amsterdam, The Netherlands

PRENTICE-HALL, INC.

ENGLEWOOD CLIFFS, NEW JERSEY

Library of Congress Cataloging in Publication Data

TANENBAUM, ANDREW S. (date)
Structured computer organization.

Bibliography
1. Electronic digital computers—Programming.
I. Title.
QA76.6.T38 001.6'42 74-30322
ISBN 0-13-854505-7

10 9 8 7

Printed in the United States of America

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA, PTY. LTD., *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA (PTE.) LTD., *Singapore*

4. The address map would be changed to map addresses 8192 to 12,287 onto memory locations 0 to 4095.
5. Execution would continue as though nothing unusual had happened.

This technique for automatic overlaying is called **paging**, and the chunks of program read in from secondary memory are called **pages**.

A more sophisticated way of mapping addresses from the address space onto the actual memory addresses is also possible. For emphasis, we will call the addresses that the program can refer to the **virtual address space** and the actual, hard-wired memory addresses the **physical address space**. A memory map exists that relates virtual addresses to physical addresses. It is presumed that there is enough room in the secondary memory (e.g., a drum or a disk) to store the whole program and its data.

Programs are written just as though there were enough main memory for the whole virtual address space, even though such is not the case. Programs may load from or store into any word in the virtual address space, or jump to any instruction located anywhere within the virtual address space, without regard to the fact that actually there is not enough physical memory. In fact, the programmer can write his programs without even being aware that virtual memory exists. He simply thinks that the computer has a big memory.

This point is crucial and will be contrasted later with segmentation, where the programmer must be aware of the existence of segments. To emphasize it once more, paging gives the programmer the illusion that a large, continuous, linear main memory, the same size as the address space, exists when, in fact, the main memory available may be smaller than the address space. The simulation of this large main memory by paging cannot be detected by the program (except by running timing tests); whenever an address is referenced, the proper instruction or data word appears to be present. Because the programmer can program as though paging did not exist, the paging mechanism is said to be **transparent**.

The idea that a programmer may use some nonexistent feature without being concerned with how it works is not new to us, after all. The instruction set of a level 2 computer is nonexistent in the sense that none of the instructions is a hardware primitive but is, in fact, carried out by software at level 1. Similarly, the level 3 programmer can use the virtual memory without worrying about how it works. The level 2 programmers who write the operating system must, of course, know exactly how it works.

5.5.2. Implementation of Paging

One essential requirement for a virtual memory is a secondary memory in which to keep the complete program. It is conceptually simpler if one thinks of the copy of the program in the secondary memory as the original one and the pieces brought into main memory every now and then as copies, rather than the other way around.

Naturally, it is important to keep the orginal up to date. When changes are made to the copy in main memory, they should be reflected in the original (eventually), too.

The virtual address space is broken up into a number of equally sized pages. Page sizes ranging from 512 to 4096 addresses per page are common at present. The page size is always a power of 2. The physical address space is broken up into pieces in a similar way, each piece being the same size as a page, so that each piece of main memory is capable of holding exactly one page. These pieces of main memory into which the pages go are called **page frames**. In Fig. 5-23 the main memory only contains one page frame. In practical designs, it will contain anywhere from 8 to 512 or even more in a very large machine. Figure 5-24 illustrates a possible way to divide up a 64K address space.

The virtual memory of Fig. 5-24 would be implemented at level 2 by means of

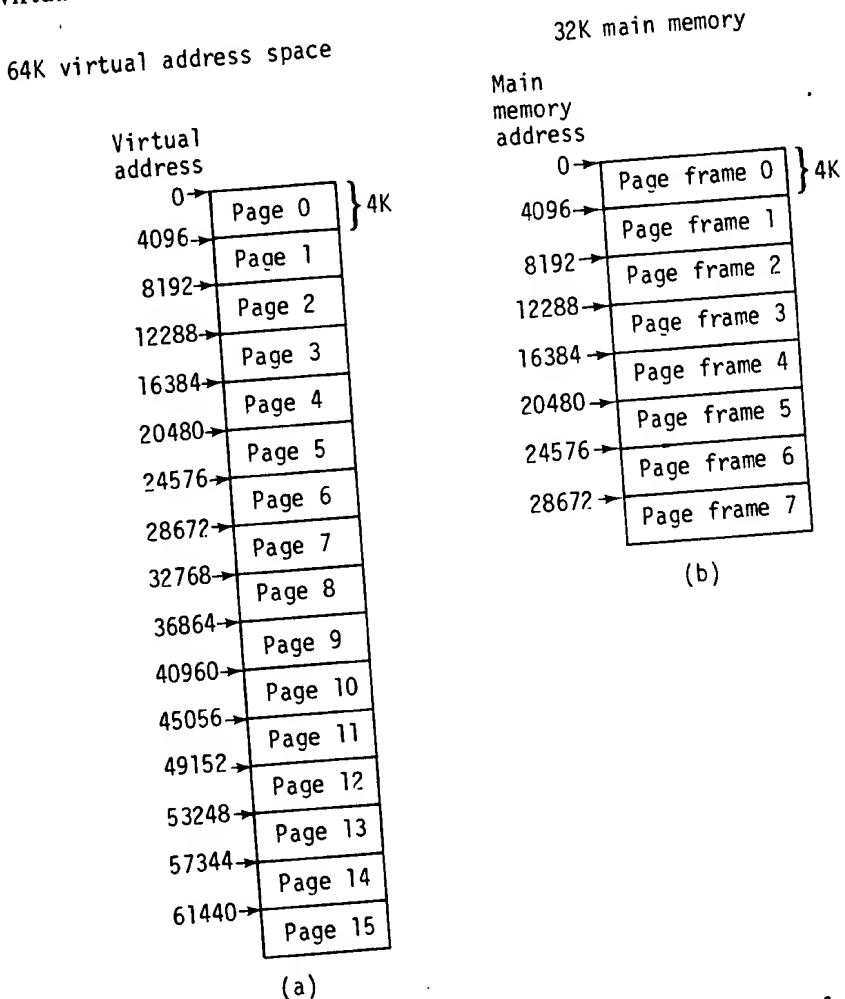

Fig. 5-24. (a) A 64K address space divided up into 16 pages of 4K each. (b) A 32K main memory divided up into eight page frames of 4K each.